13. Publishing Component Information to Embedded Software

NII52018-11.0.0

This document describes how to publish hardware component information for embedded software tools. You can publish component information for use by software, such as a C compiler and a board support package (BSP) generator. Information used by a C compiler might be a set of `#define` statements that describe some aspect of a component. Information used by a BSP generator might be the identification of memory components, so that the BSP generator can create a linker script.

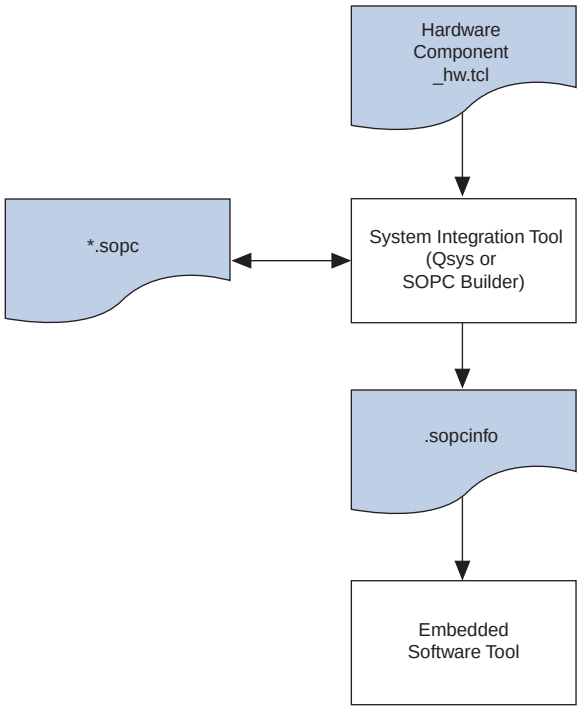
This chapter contains the following sections:

- "Embedded Component Information Flow" on page 13–1
- "Embedded Software Assignments" on page 13–2

Embedded Component Information Flow

Figure 13–1 shows the flow of information from hardware components to embedded software tools.

Figure 13–1. Embedded Component Information Flow



© 2011 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX are Reg. U.S. Pat. & Tm. Off. and/or trademarks of Altera Corporation in the U.S. and other countries. All other trademarks and service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

A component publishes information by including Tcl assignment statements in its component description file, <*component_name*>**_hw.tcl**. Each assignment is a name-value pair that can be associated with the entire component, or with a single interface. When the assignment statement applies to the entire component, it is set using the `set_module_assignment` command. Assignment statements that apply to an interface are set using the `set_interface_assignment` command. Example 13–1 shows the syntax for these assignment statements.

Example 13–1. Syntax of Assignment Statements

```
# These assignments apply to the entire component
# This is the syntax for the set_module_assignment command:
#   set_module_assignment <assignment_name> <value>

# Here are 3 examples
set_module_assignment embeddedsw.CMacro.colorSpace "CMYK"
set_module_assignment embeddedsw.configuration.cpuArchitecture "My processor"
set_module_assignment embeddedsw.memoryInfo.IS_FLASH 1


# This is the syntax of the set_interface_assignment command:
#   set_interface_assignment <interface_name> <assignment_name> <value>

# Here is an example
set_interface_assignment lcd0 embeddedsw.configuration.isPrintableDevice 1
```

For more information about the **_hw.tcl** file and using Tcl to define hardware components, refer to the *System Design with Qsys* section in *Volume 1: Design and Synthesis* of the *Quartus® II Handbook*.

When you generate a hardware system, the system integration tool, Qsys or SOPC Builder, creates an <*sopc_builder_system*>**.sopcinfo** file that includes all of the assignments for your component. The embedded software tools use these assignments for further processing. The system integration tool does not require any of the information included in these assignments to build the hardware representation of the component. The tool simply passes the assignments from the **_hw.tcl** file to the SOPC Information File (**.sopcinfo**).

Embedded Software Assignments

Embedded software assignments are organized in a period-separated namespace. All of the assignments for embedded software tools have the prefix `embeddedsw.` The `embeddedsw` namespace is further divided into the following three sub-namespaces:

- C Macro—Assignment name prefix `embeddedsw.CMacro`
- Configuration—Assignment name prefix `embeddedsw.configuration`
- Memory Initialization—Assignment name prefix `embeddedsw.memoryInfo`

C Macro Namespace

You can use the C macro namespace to publish information about your component that is converted to `#define`'s in a C or C++ `system.h` file. C macro assignments are associated with the entire hardware component, not with individual interfaces.

The name of an assignment in the C macro namespace is `embeddedsw.CMacro.<assignmentName>`. You must format the value as a legal C or C++ expression.

Example 13–2 illustrates a Tcl assignment statement for the `BAUD_RATE` of uart_0 in a hardware system.

Example 13–2. C Macro Example

```
# Tcl assignment statement included in the _hw.tcl file
add_parameter BAUD_RATE_PARAM integer 9600 "This is the default baud rate."

# Dynamically reassign the baud rate based on the parameter value
set_module_assignment embeddedsw.CMacro.BAUD_RATE \
    [get_parameter_value BAUD_RATE_PARAM]
```

Example 13–3 illustrates the corresponding C or C++ `#define`. The string `BAUD_RATE` is appended to the name of the component. This `#define` is included in the `system.h` file.

Example 13–3. Generated Macro in system.h

```
/* Generated macro in the system.h file after dynamic reassignment */
#define UART_0_BAUD_RATE 15200
```

Table 13–1 provides examples of how to format constants for 32-bit processors using the GNU Compiler Collection (GCC) C/C++ compiler.

For complete details on formatting constants, refer to the GNU web page.

Table 13–1. GCC C/C++ 32-bit Processor Constants

C Data Type	Examples
boolean (`char`, `short`, `int`)	`1, 0`
32-bit signed integer (`int`, `long`)	`123, -50`
32-bit unsigned integer (`unsigned int`, `unsigned long`)	`123u, 0xef8472a0`
64-bit signed integer (`long long int`)	`4294967296LL, -4294967296LL`
64-bit unsigned integer (`unsigned long long int`)	`4294967296ULL, 0xac458701fd64ULL`
32-bit floating-point (`float`)	`3.14f`
64-bit floating-point (`double`)	`2.78, 314e-2`
character (`char`)	`'x'`
string (`const char*`)	`"Hello World!"`

Configuration Namespace

You can use the configuration namespace to pass configuration information to embedded software tools. You can associate configuration namespace assignments with the entire component or with a single interface.

The assignment name for the configuration namespace is `embeddedsw.configuration.<name>`. Altera's embedded software tools already have definitions for the data types of the configuration names listed in this section.

Table 13–2 shows how to format configuration assignment values based on defined data types.

Table 13–2. Configuration Data Types

Configuration Data Type	Format
`boolean`	1, 0
`32-bit integer`	123, -50
`64-bit integer`	4294967296, -4294967296
`32-bit floating-point`	3.14
`64-bit floating-point`	2.78, 314e-2
`string`	ABC

Table 13–3 includes the embedded software configuration names that apply to the entire component.

Table 13–3. Component Configuration Information - Assign with set_module_assignment

Configuration Name	Type	Default	Meaning	Example
`cpuArchitecture`	string	—	Processor instruction set architecture. Provide this assignment if you want your component to be considered a processor.	My 8051
`requiredDriver`	boolean	0	If this configuration is 1 (true), the component requires a software driver. Software tools are expected to generate a warning if no driver is found.	1

Table 13–4 includes the embedded software configuration names that apply to an Avalon Memory-Mapped® (Avalon-MM) slave interface. All of these assignments are optional.

Table 13–4. Memory-Mapped Slave Information - Assign with set_interface_assignment (Part 1 of 2)

Configuration Name	Type	Default	Meaning	Examples
`isMemoryDevice`	boolean	0	The slave port provides access to a memory device.	Altera® On-Chip Memory Component, DDR Controller, erasable programmable configurable serial (EPCS) Controller
`isPrintableDevice`	boolean	0	The slave port provides access to a character-based device.	Altera UART, Altera JTAG UART, Altera LCD

Table 13–4. Memory-Mapped Slave Information - Assign with set_interface_assignment (Part 2 of 2)

Configuration Name	Type	Default	Meaning	Examples
`isTimerDevice`	boolean	0	The slave port provides access to a timer device.	Altera Timer
`isEthernetMacDevice`	boolean	0	The slave port provides access to an Ethernet media access control (MAC).	Altera Triple-Speed Ethernet
`isNonVolatileStorage` *(1)*	boolean	0	The memory device is a non-volatile memory device. The contents of a non-volatile memory device are fixed and always present. In normal operation, you can only read from this memory. If this property is true, you must also set `isMemoryDevice` to true.	Common flash interface (CFI) Flash, EPCS Flash, on-chip FPGA memory configured as a ROM
`isFlash`	boolean	0	The memory device is a flash memory device. If `isFlash` is true, you must also set `isMemoryDevice` and `isNonVolatileStorage` to true.	CFI Flash, EPCS Flash
`hideDevice`	boolean	0	Do not make this slave port visible to the embedded software tools.	Nios® II debug slave port
`affectsTransactionsOnMasters`	string	empty string	A list of master names delimited by spaces, for example `m1 m2`. Used when the slave port provides access to Avalon-MM control registers in the component. The control registers control transfers on the specified master ports. The slave port can configure the control registers for master ports on the listed components. The address space for this slave port is composed of the address spaces of the named master ports. Nios II embedded software tools use this information to generate `#define` directives describing the address space of these master ports.	Altera direct memory access (DMA), Altera Scatter/Gather DMA

Note to Table 13–4:

(1) Some FPGA RAMs support initialization at power-up from the SRAM Object File (**.sof)** or programmer object file (.**pof**), but are not considered non-volatile because this ability might not be used.

Table 13–5 includes the embedded software configuration names that apply to an Avalon Streaming® (Avalon-ST) slave interface. All of these assignments are optional.

Table 13–5. Streaming Source Information - Assign with set_interface_assignment

Configuration Name	Type	Default	Meaning	Examples
`isInterruptControllerSender` *(1)*	boolean	0	The interface sends interrupts to an interrupt controller receiver interface.	Altera Vectored Interrupt Controller
`transportsInterruptsFromReceivers` *(2)*	string	empty string	A list of interrupt receiver interface names delimited by spaces. Used when the interrupt controller sender interface can transport daisy-chained interrupts from one or more interrupt controller receiver ports on the same module.	Altera Vectored Interrupt Controller daisy-chain input

Note to Table 13–5:

(1) An interrupt sender interface is an Avalon-ST source providing interrupt information according to the external interrupt controller (EIC) protocol.

(2) An interrupt receiver interface is an Avalon-ST sink receiving interrupt information from an EIC.

Table 13–6 includes the embedded software configuration names that apply to an Avalon-ST sink interface. All of these assignments are optional.

Table 13–6. Streaming Sink Information - Assign with set_interface_assignment

Configuration Name	Type	Default	Meaning	Examples
`isInterruptControllerReceiver` *(1)*	boolean	0	The interface receives interrupts (optionally daisy-chained) from an interrupt controller sender interface.	Altera Vectored Interrupt Controller, Altera Nios II

Note to Table 13–6:

(1) An interrupt receiver interface is an Avalon-ST sink receiving interrupt information from an EIC.

Memory Initialization Namespace

You use the memory initialization namespace to pass memory initialization information to embedded software tools. Use this namespace to create memory initialization files, including **.flash**, **.hex**, **.dat**, and **.sym** files. You use memory initialization files for the following tasks:

- Flash programming
- RTL simulation
- Creating initialized FPGA RAMs for Quartus II compilation

You only need to provide these assignments if your component is a memory device that you want to initialize.

The assignment name for the memory initialization namespace is `embeddedsw.memoryInfo.<name>`. Altera® embedded software tools already have definitions for the data types of the possible values. Table 13–7 shows how to format memory initialization assignment values for all defined data types.

Table 13–7. Memory Initialization Data Types

Memory Initialization Data Type	Format
`boolean`	1, 0
`32-bit integer`	123, -50
`string` *(1)*	ABC

Note to Table 13–7:

(1) Quotation marks are not required.

Memory initialization assignments are associated with an entire component. Table 13–8 shows the embedded software memory initialization names.

Table 13–8. Memory Initialization Information - Assign with set_module_assignment Command

Memory Initialization Name	Type	Default	Meaning
`HAS_BYTE_LANE`	`boolean`	0	Create a memory initialization file for each byte.
`IS_FLASH`	`boolean`	0	Component is a flash device.
`IS_EPCS`	`boolean`	0	If `IS_FLASH` and `IS_EPCS` are both 1, component is an EPCS flash device. If `IS_FLASH` is 1 and `IS_EPCS` is 0, the component is a CFI flash device. If `IS_EPCS` is 1, `IS_FLASH` must also be 1.
`GENERATE_HEX`	`boolean`	0	Create an Intel hexadecimal file (**.hex**).
`GENERATE_DAT_SYM`	`boolean`	0	Create a .**dat** and a .**sym** file.
`GENERATE_FLASH`	`boolean`	0	Create a Motorola S-record File (**.flash**).
`INCLUDE_WARNING_MSG`	`string`	empty string	Display a warning message when creating memory initialization files.
`MEM_INIT_FILENAME`	`string`	Module instance name	Name of the memory initialization file, without any file type suffix.
`MEM_INIT_DATA_WIDTH`	`32-bit integer`	none (mandatory)	Width of memory initialization file in bits. May be different than the slave port data width.

Document Revision History

Table 13–9 shows the revision history for this document.

Table 13–9. Document Revision History (Part 1 of 2)

Date	Version	Changes
May 2011	11.0.0	Introduction of Qsys system integration tool
February 2011	10.1.0	Removed “Referenced Documents” section.
July 2010	10.0.0	Maintenance release.

Table 13–9. Document Revision History (Part 2 of 2)

Date	Version	Changes
November 2009	9.1.0	Add the following Avalon-ST interface settings, to support external interrupt controllers: ■ `embeddedsw.configuration.isInterruptControllerReceiver` ■ `embeddedsw.configuration.isInterruptControllerSender` ■ `embeddedsw.configuration.transportsInterruptsFromReceivers`
March 2009	9.0.0	Initial release.